ENGR-UH 3511

Computer Organization and Architecture

Name:	Nishant Aswani
Net ID:	nsa325
Assignment Title:	Homework 6

ENGR-UH 3511 Homework 6

Nishant Aswani (nsa325@nyu.edu)

October 29, 2019

Question 5.2.1

Given the following 32-bit memory addresses:

3, 180, 43, 2, 191, 88, 190, 14, 181, 44, 186, 253

The cache size is 16 blocks, so n=4 bits are used for the index ($2^4 = 16$).

The block size is 1 word, so m=0 bits are used for the word within the block.

Tag field size = 32 - (4 + 0 + 2) = 26

Word Address	Binary Address (preceded by 24 0s)	Tag (preceded by 22 0s)	Index	Hit/Miss
3	00000011	0000	0011	Miss
180	10110100	1011	0100	Miss
43	00101011	0010	1011	Miss
2	00000010	0000	0010	Miss
191	10111111	1011	1111	Miss
88	01011000	0101	1000	Miss
190	10111110	1011	1110	Miss
14	00001110	0000	1110	Miss
181	10110101	1011	0101	Miss
44	00101100	0010	1100	Miss
186	10111010	1011	1010	Miss
253	11111101	1111	1101	Miss

Question 5.2.2

The cache size is 8 blocks, so n=3 bits are used for the index ($2^3 = 8$).

The block size is 2 words, so m=1 bit is used for the word within the block.

Tag field size = 32 - (3 + 1 + 2) = 26

Word Address	Binary Address (preceded by 24 0s)	Tag (preceded by 22 0s)	Index	Hit/Miss
3	00000011	0000	001	Miss
180	10110100	1011	010	Miss
43	00101011	0010	101	Miss
2	00000010	0000	001	Hit
191	10111111	1011	111	Miss
88	01011000	0101	100	Miss
190	10111110	1011	111	Hit
14	00001110	0000	111	Miss
181	10110101	1011	010	Hit
44	00101100	0010	110	Miss
186	10111010	1011	101	Miss
253	11111101	1111	110	Miss

Question 5.2.3

C1 has 8 blocks, each 1 word; index = 3 bits; offset = 0 bits;
C2 has 4 blocks, each 2 words; index = 2 bits; offset = 1 bit;
C3 has 2 blocks, each 4 words; index = 1 bit; offset = 2 bits;

WA	Bin. Add.	Tag_{dec}	Index (C1)	H/M (C1)	Index (C2)	H/M (C1)	Index (C3)	H/M (C3)
3	00000011	0	011	M	01	M	0	M
180	10110100	22	100	M	10	M	1	M
43	00101011	5	011	M	01	M	0	M
2	00000010	0	010	M	01	H	0	H
191	10111111	23	111	M	11	M	1	M
88	01011000	11	000	M	00	M	0	M
190	10111110	23	110	M	11	H	1	H
14	00001110	1	110	M	11	M	1	M
181	10110101	22	101	M	10	M	1	H
44	00101100	5	100	M	10	H	1	M
186	10111010	23	010	M	01	M	0	M
253	11111101	31	101	M	10	M	1	M

C1 has a miss rate of 100%; 12(2) + 12(25) = 324 cycles
C2 has a miss rate of 75%; 12(3) + 9(25) = 261 cycles
C3 has a miss rate of 75%; 12(5) + 9(25) = 285 cycles

Cache 2 has the lowest cycle count and thus performs the best in this account

Question 5.2.4

Assume that 4 bytes = 1 word.

We can derive that the cache size is 32 KiB = 32768 B = 8192 words, assuming 32-bit addresses.

We can also derive that with a block size of 2 words, there are 4096 blocks. This implies we need 12 bits for the index field and 1 bit for the word offset. Using the formula for tag size:

32 - (12 + 1 + 2) = 17 bits for tag size

For direct mapping, we also require 1 valid bit.

Total number of bits = 4096 (64 + 17 + 1)

Total number of bits = 335872 bits

If we want to design a new cache with 16 word blocks with at least 33587 bits, we have to solve the following:

335872 <= blocks (blockSize + tagSize + validBitSize)

We know that validBitSize = 1, and that m=4, so $2^{4+2} * 8$ bytes is the blockSize

335872 <= blocks (512 + tagSize + 1)

We can rewrite the problem as:

$335872 <= 2^n(512 + [32 - (n + 4 + 2)] + 1)$

$335872 <= 2^n(539 - n)$

$n >= 9.39 \cong 10$ (we must round up)

This gives us a block size of 1024, or a total size of 67712 bytes.

For the second cache, although there may be a lower miss rate because each cycle access more data, there is a much larger penalty associated with each miss. The second cache will result in a longer access time since 16 blocks are being accessed per word, indicating that a miss is time-costly.

Question 5.2.5

Assumptions: the direct-mapping will use the modulo method to get the cache block for a given memory address. If needed, we also assume that 4 bytes = 1 word. Also, the associative cache uses the least recently used replacement policy.

Direct mapping can have a high miss rate if we constantly read from addresses that map to the same block in the cache. For example, using the 32 KiB cache with a block size of 2 words, we have 4096 blocks. Hence, we would modulo the address with 4096.

0 % 4096 = 0

4096 % 4096 = 0

If we constantly alternated between these addresses, then every single read would be a miss because of the overwrite.

On the other hand, using a 2-way associative 2 KiB cache with a block size of 1 word, we have 512 blocks.

2-way associativity implies there would be 256 sets. Hence, we would modulo the address with 256.

0 % 256 = 0

4096 % 256 = 0

Since this is a set of two blocks, there are only two misses (first two reads). Beyond then, the remaining reads will all be hits.

Hence, the reads can be: 0, 4096, 0, 4096, 0, 4096,...

An advantage is that even with a much lower capacity, the system is quite robust. It is able to handle the case of same mapping quite well because it can store addresses with the same index. Hence, there is a lower miss rate. On the other hand, there is a higher hit time with the two-way set associative cache. Not only are we no comparing to check the index, but we must look within the set to find the correct block as well. This also requires extra hardware (such as comparators). Another disadvantage for this specific two-way associative cache may be that if read requests looked like:

0, 4096, 8192, 0, 4096, 8192, ...

then the miss rate would be equally bad in both cases. This is because the third read is a miss, but it also replaces the block from memory address 0 under the LRU policy. The next read request is block 0 which obviously is a miss.